David Chappell

REDEFINING QUALITY ASSURANCE

AN ALM PERSPECTIVE

Sponsored by Microsoft Corporation

Here's a simple way for your organization to increase the quality of the software it creates: Stop saying "QA" when you mean "testing". If you have a quality assurance group that actually does testing, change the group's name. Every time you hear one of your co-workers use the term "QA" to mean "testing", correct him. And whenever you find yourself saying "QA" in your head when you're really thinking about testing, stop and set yourself straight.

The reason for all of this is that correct thinking is the best path to correct action, and equating QA with testing is far from correct. Testing is important, but achieving real software quality requires a much broader view. A better approach is to look at quality through the wider lens of application lifecycle management (ALM). Once you do this, it becomes obvious that quality assurance is part of every aspect of ALM.

The Traditional View of Quality Assurance: It's Testing

Whether you're using waterfall or agile development, the traditional view of QA focuses on the testing part of the process. Figure 1 illustrates this idea.

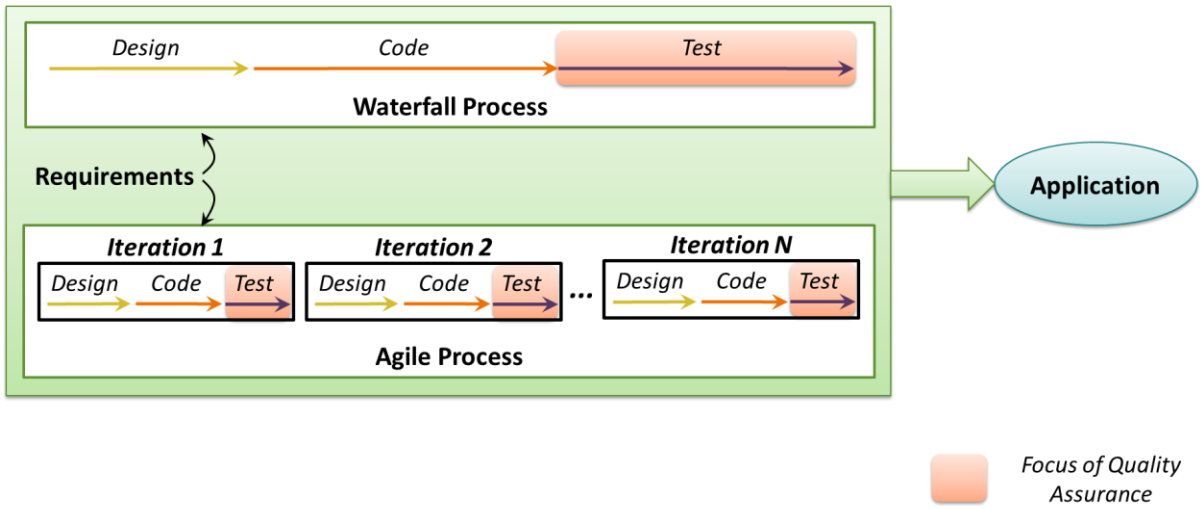

Figure 1: In both waterfall and agile development, the traditional view equates QA with testing.

Both waterfall and agile processes turn requirements into an application, and both rely on the basic functions of design, code, and test. Organizations that use waterfall processes often call their testing group "QA", and even though agile processes tend to spread the testing responsibility more broadly, it's not uncommon to use the same terminology: QA equals testing.

Creating high-quality software requires much more than just good testing.

But we all know that this isn't true. Creating high-quality software requires much more than just good testing. To really assure quality, we need the broader perspective that ALM provides.

An ALM View of Quality Assurance

To look at QA from an ALM perspective, we first need to understand what ALM is. A simple way to think about it is to divide ALM into three parts[1], as Figure 2 shows.

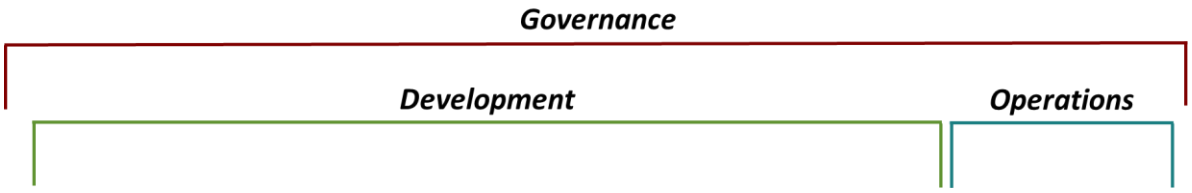

Figure 2: Application lifecycle management can be thought of as having three parts.

The three aspects of ALM are:

- *Development*, the process of creating an application. This aspect of ALM is sometimes referred to as the *software development lifecycle (SDLC)*.
- *Operations*, which entails running and managing the application after it's deployed. Many people think of ALM as equivalent to SDLC, but this isn't correct. ALM encompasses an application's entire lifecycle, from the time someone first has the idea to create it all the way until the application is removed from service. Operations is a critically important part of this, and so it's also a fundamental part of ALM.
- *Governance*, which includes all of the decisions made about the application during its lifetime. This includes creating the business case for the application, project management during development, and deciding when the application should be taken out of service. As Figure 2 shows, governance spans both operations and development.

This simple structure gives us a more accurate way to think about what QA really means. Visualizing this is easy; we just need to combine Figures 1 and 2. Figure 3 shows how this looks.

[1] For a more detailed look at this approach, see What is Application Lifecycle Management?, David Chappell.

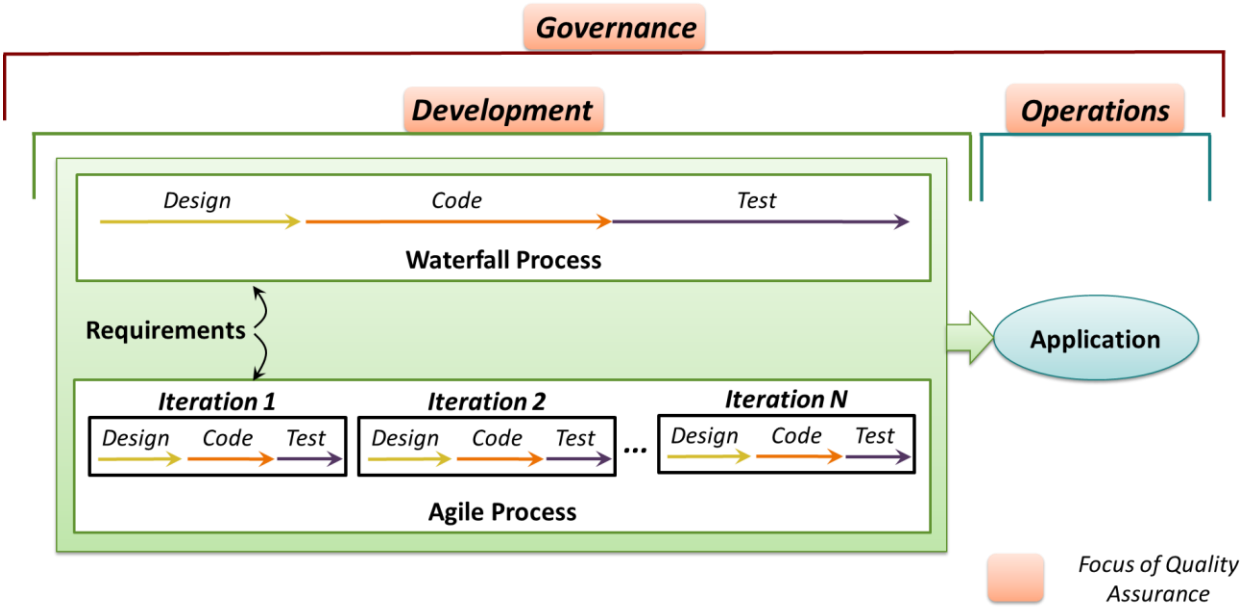

Figure 3: An ALM perspective on QA recognizes that every aspect of the ALM process plays a role in ensuring software quality.

The truth is that assuring quality requires focusing on every aspect of ALM: Development, Operations, and Governance. Testing is part of this, but it's not even the most important part. And there certainly can't be a distinct group called "QA", because ensuring software quality is part of everybody's job.

It's worth taking a closer look at how quality fits into each of ALM's three aspects. For Development, the important areas include these:

- Requirements: Quality begins with defining the right product. A bug-free application that doesn't solve the right problem is still low-quality software. To solve the right problem, you need high-quality requirements. Choosing which users supply them is important, as is the process used to capture and describe those requirements. (One of the most important benefits of agile processes over older waterfall approaches is that agile's iterative structure is better at handling additions and changes to requirements while the application is being created.)

- Design: Any good software architect knows that overall software quality is rooted in quality design. Many aspects of software quality, including performance and testability, depend on good design. Even operational issues, such as manageability and architecting for low cost when running in a public cloud, depend on high-quality design.

- Code: Over the last several decades, our industry has created many coding practices that help increase software quality: unit testing, continuous integration with build verification tests, test-driven development, methodologies for doing security reviews, and lots more. The developers writing the code for an application clearly play a large part in quality assurance.

- Test: While it's not the whole story, the traditional home of QA remains an important part of providing software quality. Manual exploratory testing, automated testing, and other approaches are fundamental to making sure that code does what it's supposed to. But testing by itself can't provide software quality, if only because it comes too late in the process. The best it can do is find the places where earlier efforts to ensure quality have failed.

The second part of ALM, Operations, also impacts overall quality. Among the most important of these effects are the following:

- Reliability: To its users, an application that isn't available when they need it isn't high quality. But whether this unavailability is due to internal bugs or poor operations doesn't matter to them—it's still a low-quality application. Focusing on code quality, working hard to squash every bug during development, without also focusing on reliable operation of the application makes no sense. A development team that creates a great piece of software can see its hard work wasted by unreliable operations. Especially as more applications use the Software as a Service (SaaS) approach, high-quality code means nothing if it's not paired with reliable operations.

- Security: Application security depends on good code, but effective operations also plays an important role. Leaving the wrong port open in a firewall, for example, something that's usually outside the control of the development team, can open an application to unexpected attacks. And without good security, no application has high quality.

Governance, the third aspect of ALM, is also critical to quality. Here are some examples:

- Initiation: The most important decision made on any development project is the first one: Is this application worth building? Spending money creating custom software when an equivalent (or maybe even better) product is available off the shelf is definitely not a high-quality decision. Creating custom code that addresses low-priority business problems also isn't a high quality choice. Making the wrong decision here is an effective way to waste a significant amount of money.

- Schedule: How many development groups cut corners on quality because of schedule pressure? Delivering new software quickly can have real business value, and so trade-offs here are unavoidable. Still, schedule decisions are an aspect of governance, and so thinking about quality assurance without considering this aspect of ALM makes no sense.

- End of life: Software quality isn't just about bugs; it's also about getting the most for your money. An important part of this is deciding when an application has reached the end of its useful life and can be taken out of service. This can be hard in many organizations, since there's always somebody who still uses the application. Yet making this decision well can save significant amounts of money, and it's another example of quality in the ALM process.

Quality can't be tested in; it must be built in throughout a project.

It's become a cliché, but it remains true: Quality is everybody's job. Quality can't be tested in—it must be built in throughout a project—and so passing the quality buck to a QA team doesn't work. Software quality comes from every part of the ALM process, and so everybody needs to feel responsible for it.

Conclusion

In our hearts, we all know that QA is more than testing; this isn't a revelation. But our perspective drives our actions—words matter—and so thinking clearly requires using the right terminology.

We also know that perfection is never possible. Quality in software really means achieving an acceptable level of risk. Yet whatever that level is, no organization can create high quality software solely through testing. The best testers in the world can't stop an organization from creating low-quality software if they're part of an ineffective ALM process. What's needed is clear: a broad and holistic view of software quality that's rooted in ALM.

About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.